Church’s Thesis

Algorithms and Decidability

- Self-reference
  1. **Example**. This sentence is false.

  2. **Example**. There is a barber in a village, and he shaves precisely those men in the village who do not shave themselves. Does he shave himself?

  3. **Example**. Programs as data

- Self referencing systems are interesting because they can lead to paradoxes.

Mechanizing Mathematics

- Arithmetic and other big fragments of Mathematics can be represented as formal systems.
- Whitehead and Russel's *Principia Mathematica* (1910--1913) is a grand example of an attempt at formalizing the whole of Mathematics from the logical scratch.
- Statements are represented by precise formulas; some formulas are taken as axioms;
- Theorems are derived from axioms and previously proved theorems by precise rules of deduction.

- **Example**. $(\exists\ n \geq 3)(\exists\ x)(\exists\ y)(\exists\ z)\ x^n + y^n = z^n$

Completeness

- Some formal systems are complete:
  - all true formulas are theorems of the system.
- Examples are propositional and predicate calculi.
- In 1936, Gödel proved the famous *Incompleteness Theorem*,
- It says that the proposed formalization of Arithmetic (as well as any reasonable extension of it) is incomplete:
  - there are sentences that are true, but unprovable in the system.
- Gödel managed to express theoremhood within the system and then construct a formula that (essentially) says *I am not a theorem*.

Decision Problems

- Given a set S and a subset A of S, the corresponding *decision problem* is to find an algorithm that takes an arbitrary element x of S as input and returns True or False depending on whether $x \in A$ or $x \notin A$.

- **Example**. Is a given number n prime?

- **Example**. Is a given CFG G ambiguous?

- More generally,

- given a function $f: X \to Y$, we may be interested in the existence of an algorithm that given an input $x \in X$ produces as output $f(x) \in Y$.

- Such problems can be reduced to decision problems. Just take
- $S = X \times Y$ and
- $A = \{ (x,y) \mid y=f(x)\} \subseteq S$

- if we can solve the decision problem for A then we can compute the function f. (How?)

Languages and Decision Problems

- The decision problem for a language L over $\{0,1\}$ is to find an algorithm that, given any string $w \in \{0,1\}^*$ returns True or False, depending on whether $w \in L$ or not.

- A suitable encoding can translate any decision problem into one about a language.

Undecidable Problems Exist

- Consider the set of all algorithms (C programs, for example) that take binary strings as input and return True or False.
- Each of them accepts a language. There are infinitely many algorithms, but only *countably many*:
- We can arrange them in a sequence A_1, A_2, A_3, …
- Let L_1, L_2, L_3 … be the languages they accept.

- Now for each language $L_{i,}$ write $L_i=\{w_{i1},w_{i2},w_{i3}, \ldots \}$, ordering the elements lexicographically.

- Now define a *new* language L (different from all the L_i).
- The language L is defined inductively by:
- L= {w1,w2,w3,…}. We require
- w_1 to be greater than w_{11},
- w_2 to be greater than both w_1 and w_{22},
- w_3 to be greater than both w_2 and w_{33}
- etc.

Diagonalization

- Clearly, the elements of L are lexicographically ordered (since $_{wi+1}$ is taken to be greater than w_i). Since the i^{th} element of L is greater than the i^{th} element of L_i, it follows that $L \neq L_i$, for all i. Thus, no algorithm recognizes the language L.

- Constructions such as this use *the diagonalization argument*. It was first used by Cantor to show that the set of real numbers cannot be put into one-to-one correspondence with the set of natural numbers, and so is *uncountable*.

Undecidable Problems in concrete terms

- Consider all pairs (P,I), where P is a C program of type String $\rightarrow$ String and I is a string.
- Let L be the set of all pairs (P,I) such that P(I)= "`hello`" .
- We claim that there exist no algorithm (C program) for the decision problem of L (the set of all pairs (P,I)).
- Assume the contrary. Then there exists a program H such that:

$$H(P,I) = \begin{cases} \text{No, if } P(I) \neq \text{“}\texttt{hello}\text{”} \\ \\ \text{Yes, if } P(I) = \text{“}\texttt{hello}\text{”} \end{cases}$$

- Let H_1 be the program defined by
  - H_1(P,I) = if H(P,I) = no then "`hello`" else "`yes`"
- Finally, let H_2 be the program defined by
  - $H_2(P) = H_1(P,P)$
- What is $H_2(H_2)$ now?

  - If $H_2(H_2)$= "`yes`" then (the definition of H_1) implies $H(H_2,H_2)$= "`yes`" , and then (the definition of H) implies $H_2(H_2)$= "`hello`" .

  - If $H_2(H_2)$= "`hello`" then (the definition of H_1) implies $H(H_2,H_2)$= no, and then (the definition of H) implies $H_2(H_2) \neq$ "`hello`".

- Both can't be true, so we have a Contradiction, Which means our original assumption that H must exist was flawed.

Undecidable Problems all around us?

- No matter how hard you try, you'll never manage to write a C program that tests a CFG grammar for ambiguity. That's undecidable, as well as many other problems about grammars.

- We'd like to know more now about undecidable problems, just to get some idea of what is impossible to program.

What is an Algorithm?

- To prove that a certain problem is undecidable, we have to show that there is no algorithm for it.
- For such a proof, however, we need a precise definition of *algorithm*.
- In the above example, we identified algorithms with C programs.

- But is that OK? Is it true that every problem that has an algorithmic solution also has a solution by a C program?

- Even if the answer to the last question is “yes”, it does not seem right to define algorithms as C programs. A mathematical definition must be simpler!

Computable Functions

- Importance of having precise definitions of *effectively* computable functions, or algorithms, was understood in the 1920's. There were several attempts to formalize the basic notions of computability:

  Turing Machines
  Post Systems
  Recursive Functions
  Markov Algorithms
  λ-calculus

  - We will study many of these in the next few days

- On the surface, these approaches look quite different. It turned out, however, that they are all equivalent! All these, and all later formalizations (combinatory logic, *while* programs, C programs, etc.) give essentially the same meaning to the word *algorithm* .

Church’s Thesis

- The statement that these formalizations correspond to the intuitive concept of computability is known as *Church's Thesis.*

- Church's Thesis is a belief, not a theorem.

- (though we often act as if we believe it is true, even though we don’t know its is true)